ASSIGNMENT 4:

EE599

Submitted by: Charu Singh

I have used Mobilenet_v2 pretrained model for this assignment as the baseline.

Test accuracy of Mobilenet_v2 is better than my custom model.

Advantages of using pretrained model:

It allows us to build accurate models in compartively less time. In custom model we learn all tye features from the scratch, we have to learn all the features which are already learned in solving a different problem this is why using pretrained model allows us to avoid the learning process from scratch.

In order to reduce the computational cost of traning process we use pretrained models.

Final dense layers in our network allows us to identufy spoecific characterstic of our dataset that needs traing. Therfore, we take the baseline model that is mobilenet_v2 and add some dense layers and then train our new finetuned model.

Compatibilty:

In order to create new dataset for this task I created all possible combinations of the images for a particular set ID which I am extracting from train.json file.

Results:

Accuracy achieved for *categorical fine tuned* model is 66% and custom model is 63% for 20 epochs.

Accuracy achieved for *compatibilty model* is 63% for 10 epochs.

I have included accuracy and loss graphs in separate file.

Bonus Tasks:

I have included learning rate scheduling and data augmentation in *get_data_transform* function to increase the robustness of the model.

CODE:

1. **FINE TUNED MODEL:**

```
#!/usr/bin/env python
# coding: utf-8

# In[4]:

# utils.py

import numpy as np
import os
import os.path as osp
import argparse

Config ={}
Config['root_path'] = "./polyvore_outfits/"
Config['meta_file'] = "polyvore_item_metadata.json"
Config['test_file'] = "test_category_hw.txt"
Config['checkpoint_path'] = ''
#Config['train_compatibility']='pairwise_compatibility_train.txt'
#Config['valid_compatibility']='pairwise_compatibility_valid.txt'


Config['use_cuda'] = True
Config['debug'] = False
Config['num_epochs'] = 20
Config['batch_size'] = 64

Config['learning_rate'] = 0.001
Config['num_workers'] = 5 #aws might not need it, original value = 5

# In[2]:

# data.py

import torch
import torch as th
```

```
import torch.nn as nn
import torch.nn.functional as F
from torchvision import transforms
from torch.utils.data import Dataset, DataLoader

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder

import os
import numpy as np
import os.path as osp
import json
from tqdm import tqdm
from PIL import Image

#from utils import Config

class polyvore_dataset:
    def __init__(self):
        self.root_dir = Config['root_path']
        self.image_dir = osp.join(self.root_dir, 'images')
        self.transforms = self.get_data_transforms()
        # self.X_train, self.X_test, self.y_train, self.y_test, self.classes = self.create_dataset()


    def get_data_transforms(self):
        data_transforms = {
            'train': transforms.Compose([
                transforms.CenterCrop(224),
                transforms.ToTensor(),
                transforms.Normalize([0.5, 0.5, 0.5], [0.5, 0.5, 0.5])
            ]),
            'test': transforms.Compose([
                transforms.Resize(256),
                transforms.CenterCrop(224),
                transforms.ToTensor(),
                transforms.Normalize([0.5, 0.5, 0.5], [0.5, 0.5, 0.5])
            ]),
        }
        return data_transforms


    def create_dataset(self):
        # map id to category
```

```
        meta_file = open(osp.join(self.root_dir, Config['meta_file']), 'r'
    )
        meta_json = json.load(meta_file)
        id_to_category = {}
        for k, v in tqdm(meta_json.items()):
            id_to_category[k] = v['category_id']

        # create X, y pairs
        files = os.listdir(self.image_dir)
        X = []; y = []
        for x in files:
            if x[:-4] in id_to_category:
                X.append(x)
                y.append(int(id_to_category[x[:-4]]))

        y = LabelEncoder().fit_transform(y)
        print('len of X: {}, # of categories: {}'.format(len(X), max(y) +
    1))

        # split dataset
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_siz
    e=0.2)
        return X_train, X_test, y_train, y_test, max(y) + 1


# For category classification
    class polyvore_train(Dataset):
        def __init__(self, X_train, y_train, transform):
            self.X_train = X_train
            self.y_train = y_train
            self.transform = transform
            self.image_dir = osp.join(Config['root_path'], 'images')

        def __len__(self):
            return len(self.X_train)

        def __getitem__(self, item):
            file_path = osp.join(self.image_dir, self.X_train[item])
            return self.transform(Image.open(file_path)),self.y_train[it
    em]



    class polyvore_test(Dataset):
        def __init__(self, X_test, y_test, transform):
```

```
116.             self.X_test = X_test
117.             self.y_test = y_test
118.             self.transform = transform
119.             self.image_dir = osp.join(Config['root_path'], 'images')
120.
121.         def __len__(self):
122.             return len(self.X_test)
123.
124.         def __getitem__(self, item):
125.             file_path = osp.join(self.image_dir, self.X_test[item])
126.             return self.transform(Image.open(file_path)), self.y_test[it
    em]
127.
128.
129.     def get_dataloader(debug, batch_size, num_workers):
130.         dataset = polyvore_dataset()
131.         transforms = dataset.get_data_transforms()
132.         X_train, X_test, y_train, y_test, classes = dataset.create_datas
    et()
133.
134.         if debug==True:
135.             train_set = polyvore_train(X_train[:100], y_train[:100], tra
    nsform=transforms['train'])
136.             test_set = polyvore_test(X_test[:100], y_test[:100], transfo
    rm=transforms['test'])
137.             dataset_size = {'train': len(y_train), 'test': len(y_test)}
138.         else:
139.             train_set = polyvore_train(X_train, y_train, transforms['tra
    in'])
140.             test_set = polyvore_test(X_test, y_test, transforms['test'])
141.             dataset_size = {'train': len(y_train), 'test': len(y_test)}
142.
143.         datasets = {'train': train_set, 'test': test_set}
144.         dataloaders = {x: DataLoader(datasets[x],
145.                                      shuffle=True if x=='train' else Fal
    se,
146.                                      batch_size=batch_size,
147.                                      num_workers=num_workers)
148.                                      for x in ['train', 'test']}
149.         return dataloaders, classes, dataset_size
150.
151.     def text_gen():
152.             meta_file = open(osp.join(Config['root_path'], Config['meta_
    file']), 'r')
```

```
152.             meta_json = json.load(meta_file)
153.             id_to_category = {}
154.             m = 0
155.             for k, v in tqdm(meta_json.items()):
156.                 id_to_category[k] = v['category_id']
157.
158.             # create X, y pairs
159.             files = os.listdir(osp.join(Config['root_path'], 'images'))
160.             y = []
161.             for x in files:
162.                 if x[:-4] in id_to_category:
163.                     y.append(int(id_to_category[x[:-4]]))
164.             le = LabelEncoder()
165.             g = le.fit_transform(y)
166.             B = []
167.             f = open(Config['root_path'] + "test_category_hw.txt", "r")
168.             a = [line.split() for line in f.readlines()]
169.             for i in range(len(a)):
170.                 a[i][0] = a[i][0] + '.jpg'
171.                 B.append(a[i][0])
172.
173.             a = open(Config['root_path'] + "test_category_hw.txt", "r")
174.             b = open('Model_test_category_hw.txt', 'w')
175.
176.             f = [lines.split() for lines in a.readlines()]
177.             J = []
178.             for i in range(len(f)):
179.                 J.append(f[i][0])
180.
181.             dataset = polyvore_dataset()
182.             transforms = dataset.get_data_transforms()['test']
183.
184.             size = int(np.floor(len(B) / Config['batch_size']))
185.
186.             #model_copy_tensor = torch.load('./Results/ResNet.pth')
187.             check_point = torch.load('model.pth')
188.             model_copy = check_point['model']
189.             #model_copy = load_model('Build_model.hdf5')
190.             #model_copy.eval()
191.
192.             for i in range(0, size*Config['batch_size'], Config['batch_s
    ize']):
193.                 X= []
194.                 Y =[]
195.                 ans = []
```

```
                for j in range(Config['batch_size']):
                    file_path = osp.join(osp.join(Config['root_path'], '
images'), B[i+j])
                    l = transforms(Image.open(file_path))
                    X.append(l)
                    Y.append(id_to_category[J[i+j]])

                #Y = np.stack(C)
                #Y = np.moveaxis(Y, 1, 3)

                with torch.no_grad():
                    for inputs in X:
                        # print(inputs.shape)
                        inputs = inputs.to(device)
                        outputs = model_copy(inputs[None,...])
                        _, pred = torch.max(outputs,1)
                        for p in pred:
                            ans.append(p)


    #             acc, loss1, ans = eval_model(model_copy_tensor, Y, cri
terion, device)
    #             # ans = (model_copy.predict(Y))

    #             for k in range(len(ans)):
    #                 ans1.append(np.argmax(ans[k]))
                preds = le.inverse_transform(np.asarray(ans, dtype= np.i
nt32))
                for p in range(Config['batch_size']):
                    b.write(J[p+m] + '\t' + str(preds[p]) + '\t' + id_to
_category[J[p+m]] + '\n')
                m = m + Config['batch_size']
                if m == size*Config['batch_size']:
                    break
            b.close()

    ##########################################################################
####
    # For Pairwise Compatibility Classification

    # In[3]:

    # model.py

    from torchvision.models import resnet50
```

```
236.     from torchvision.models import mobilenet_v2
237.     from torch import nn
238.
239.     #model = resnet50(pretrained=True)
240.     pretrained = mobilenet_v2(pretrained=True)
241.
242.     class MyMobileNet(nn.Module):
243.         def __init__(self, my_pretrained_model):
244.             super(MyMobileNet, self).__init__()
245.             self.pretrained = my_pretrained_model
246.             self.my_new_layers = nn.Sequential(
247.                                                 nn.Dropout(0.4),
248.                                                 nn.Linear(1000, 200),
249.                                                 nn.ReLU(),
250.                                                 nn.Linear(200, 153)
251.                                                 )
252.
253.         def forward(self, x):
254.             x = self.pretrained(x)
255.             x = self.my_new_layers(x)
256.             return x
257.
258.     model = MyMobileNet(my_pretrained_model=pretrained)
259.
260.     # In[ ]:
261.
262.     # train_category.py
263.
264.     import torch
265.     import torch.nn as nn
266.     import torch.optim as optim
267.     import torch.nn.functional as F
268.
269.     import argparse
270.     import time
271.     import copy
272.     from tqdm import tqdm
273.     import os.path as osp
274.     import matplotlib.pyplot as plt
275.
276.     #from utils import Config
277.     #from model import model
278.     #from data import get_dataloader
```

```
279.
280.     def train_model(dataloader, model, criterion, optimizer, device, num
    _epochs, dataset_size):
281.         model.to(device)
282.         since = time.time()
283.         best_model_wts = copy.deepcopy(model.state_dict())
284.         best_acc = 0.0
285.         train_loss_list= []
286.         val_loss_list = []
287.         train_acc_list = []
288.         val_acc_list = []
289.
290.         for epoch in range(num_epochs):
291.             print('Epoch {}/{}'.format(epoch, num_epochs - 1))
292.             print('-' * 10)
293.
294.             for phase in ['train', 'test']:
295.                 if phase=='train':
296.                     model.train()
297.                 else:
298.                     model.eval()
299.
300.                 running_loss = 0.0
301.                 running_corrects = 0
302.
303.                 for inputs, labels in tqdm(dataloaders[phase]):
304.                     inputs = inputs.to(device)
305.                     labels = labels.to(device)
306.                     optimizer.zero_grad()
307.
308.                     with torch.set_grad_enabled(phase=='train'):
309.                         outputs = model(inputs)
310.                         _, pred = torch.max(outputs, 1)
311.                         loss = criterion(outputs, labels)
312.
313.                         if phase=='train':
314.                             loss.backward()
315.                             optimizer.step()
316.
317.                     running_loss += loss.item() * inputs.size(0)
318.                     running_corrects += torch.sum(pred==labels.data)
319.
320.                 epoch_loss = running_loss / dataset_size[phase]
```

```
                epoch_acc = running_corrects.double() / dataset_size[pha
    se]

                if phase == 'train':
                    train_loss_list.append(epoch_loss)
                    train_acc_list.append(epoch_acc)

                if phase == 'test':
                    val_loss_list.append(epoch_loss)
                    val_acc_list.append(epoch_acc)

                print('{} Loss: {:.4f} Acc: {:.4f}'.format(phase, epoch_
    loss, epoch_acc))

                if phase=='test' and epoch_acc > best_acc:
                    best_acc = epoch_acc
                    best_model_wts = copy.deepcopy(model)

            #torch.save({'model':best_model_wts}, osp.join(Config['root_
    path'], Config['checkpoint_path'], 'model.pth'))
            #print('Model saved at: {}'.format(osp.join(Config['root_pat
    h'], Config['checkpoint_path'], 'model.pth')))
            torch.save({'model':best_model_wts}, 'model.pth')
            print('Model saved at: {}'.format('model.pth'))

        time_elapsed = time.time() - since
        print('Time taken to complete training: {:0f}m {:0f}s'.format(ti
    me_elapsed // 60, time_elapsed % 60))
        print('Best acc: {:.4f}'.format(best_acc))

        plt.figure()
        plt.plot(np.arange(num_epochs),train_loss_list,label='Train')
        plt.plot(np.arange(num_epochs),val_loss_list, label='Validation'
    )
        plt.xlabel('Epoch')
        plt.ylabel('Loss')
        plt.legend()
        plt.grid()
        plt.savefig('./Resnet_new_loss.png', dpi=256)
        #plt.show()

        plt.figure()
        plt.plot(np.arange(num_epochs),train_acc_list,label='Train')
        plt.plot(np.arange(num_epochs),val_acc_list, label='Validation')
        plt.xlabel('Epoch')
```

```
360.          plt.ylabel('Accuracy')
361.          plt.legend()
362.          plt.grid()
363.          plt.savefig('./Resnet_new_acc.png', dpi=256)
364.          #plt.show()
365.
366.      if __name__=='__main__':
367.
368.          dataloaders, classes, dataset_size = get_dataloader(debug=Config
     ['debug'], batch_size=Config['batch_size'], num_workers=Config['num_worker
     s'])
369.
370.          criterion = nn.CrossEntropyLoss()
371.          optimizer = optim.RMSprop(model.parameters(), lr=Config['learnin
     g_rate'],weight_decay=0.0001)
372.          device = torch.device('cuda:0' if torch.cuda.is_available() and
     Config['use_cuda'] else 'cpu')
373.
374.          train_model(dataloaders, model, criterion, optimizer, device, nu
     m_epochs=Config['num_epochs'], dataset_size=dataset_size)
375.          text_gen()
376.
377.          print('DONE')
```

2. CUSTOM_MODEL:

```
#!/usr/bin/env python
# coding: utf-8

# In[10]:


#!/usr/bin/env python
# coding: utf-8

# In[4]:


# utils.py

import numpy as np
import os
import os.path as osp
import argparse
```

```
Config ={}
Config['root_path'] = "./polyvore_outfits/"
Config['meta_file'] = "polyvore_item_metadata.json"
Config['test_file'] = "test_category_hw.txt"
Config['checkpoint_path'] = ''
#Config['train_compatibility']='pairwise_compatibility_train.txt'
#Config['valid_compatibility']='pairwise_compatibility_valid.txt'



Config['use_cuda'] = True
Config['debug'] = False
Config['num_epochs'] = 20
Config['batch_size'] = 64

Config['learning_rate'] = 0.001
Config['num_workers'] = 5 #aws might not need it, original value = 5


# In[2]:


# data.py

import torch
import torch as th
import torch.nn as nn
import torch.nn.functional as F
from torchvision import transforms
from torch.utils.data import Dataset, DataLoader

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder

import os
import numpy as np
import os.path as osp
import json
from tqdm import tqdm
from PIL import Image

#from utils import Config


class polyvore_dataset:
```

```
    def __init__(self):
        self.root_dir = Config['root_path']
        self.image_dir = osp.join(self.root_dir, 'images')
        self.transforms = self.get_data_transforms()
        # self.X_train, self.X_test, self.y_train, self.y_test, self.classes = se
lf.create_dataset()



    def get_data_transforms(self):
        data_transforms = {
            'train': transforms.Compose([
                transforms.CenterCrop(224),
                transforms.RandomHorizontalFlip(p=0.5),
                transforms.ToTensor(),
                transforms.Normalize([0.5, 0.5, 0.5], [0.5, 0.5, 0.5])
            ]),
            'test': transforms.Compose([
                transforms.Resize(256),
                transforms.RandomHorizontalFlip(p=0.5),
                transforms.CenterCrop(224),
                transforms.ToTensor(),
                transforms.Normalize([0.5, 0.5, 0.5], [0.5, 0.5, 0.5])
            ]),
        }
        return data_transforms



    def create_dataset(self):
        # map id to category
        meta_file = open(osp.join(self.root_dir, Config['meta_file']), 'r')
        meta_json = json.load(meta_file)
        id_to_category = {}
        for k, v in tqdm(meta_json.items()):
            id_to_category[k] = v['category_id']

        # create X, y pairs
        files = os.listdir(self.image_dir)
        X = []; y = []
        for x in files:
            if x[:-4] in id_to_category:
                X.append(x)
                y.append(int(id_to_category[x[:-4]]))
```

```
        y = LabelEncoder().fit_transform(y)
        print('len of X: {}, # of categories: {}'.format(len(X), max(y) + 1))

        # split dataset
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
        return X_train, X_test, y_train, y_test, max(y) + 1



# For category classification
class polyvore_train(Dataset):
    def __init__(self, X_train, y_train, transform):
        self.X_train = X_train
        self.y_train = y_train
        self.transform = transform
        self.image_dir = osp.join(Config['root_path'], 'images')

    def __len__(self):
        return len(self.X_train)

    def __getitem__(self, item):
        file_path = osp.join(self.image_dir, self.X_train[item])
        return self.transform(Image.open(file_path)),self.y_train[item]




class polyvore_test(Dataset):
    def __init__(self, X_test, y_test, transform):
        self.X_test = X_test
        self.y_test = y_test
        self.transform = transform
        self.image_dir = osp.join(Config['root_path'], 'images')


    def __len__(self):
        return len(self.X_test)


    def __getitem__(self, item):
        file_path = osp.join(self.image_dir, self.X_test[item])
        return self.transform(Image.open(file_path)), self.y_test[item]
```

```
def get_dataloader(debug, batch_size, num_workers):
    dataset = polyvore_dataset()
    transforms = dataset.get_data_transforms()
    X_train, X_test, y_train, y_test, classes = dataset.create_dataset()

    if debug==True:
        train_set = polyvore_train(X_train[:100], y_train[:100], transform=transf
orms['train'])
        test_set = polyvore_test(X_test[:100], y_test[:100], transform=transforms
['test'])
        dataset_size = {'train': len(y_train), 'test': len(y_test)}
    else:
        train_set = polyvore_train(X_train, y_train, transforms['train'])
        test_set = polyvore_test(X_test, y_test, transforms['test'])
        dataset_size = {'train': len(y_train), 'test': len(y_test)}

    datasets = {'train': train_set, 'test': test_set}
    dataloaders = {x: DataLoader(datasets[x],
                                 shuffle=True if x=='train' else False,
                                 batch_size=batch_size,
                                 num_workers=num_workers)
                                 for x in ['train', 'test']}
    return dataloaders, classes, dataset_size


def category_text_generation():
        meta_file = open(osp.join(Config['root_path'], Config['meta_file']), 'r')
        meta_json = json.load(meta_file)
        id_to_category = {}
        m = 0
        for k, v in tqdm(meta_json.items()):
            id_to_category[k] = v['category_id']

        # create X, y pairs
        files = os.listdir(osp.join(Config['root_path'], 'images'))
        y = []
        for x in files:
            if x[:-4] in id_to_category:
                y.append(int(id_to_category[x[:-4]]))
        le = LabelEncoder()
        g = le.fit_transform(y)
        B = []
        f = open(Config['root_path'] + "test_category_hw.txt", "r")
        a = [line.split() for line in f.readlines()]
        for i in range(len(a)):
```

```
            a[i][0] = a[i][0] + '.jpg'
            B.append(a[i][0])

        a = open(Config['root_path'] + "test_category_hw.txt", "r")
        b = open('Charu_Model_test_category_hw.txt', 'w')

        f = [lines.split() for lines in a.readlines()]
        J = []
        for i in range(len(f)):
            J.append(f[i][0])


        dataset = polyvore_dataset()
        transforms = dataset.get_data_transforms()['test']

        size = int(np.floor(len(B) / Config['batch_size']))

        #model_copy_tensor = torch.load('./Results/ResNet.pth')
        check_point = torch.load('model.pth')
        model_copy = check_point['model']
        #model_copy = load_model('Build_model.hdf5')
        #model_copy.eval()

        for i in range(0, size*Config['batch_size'], Config['batch_size']):
            X= []
            Y =[]
            ans = []
            for j in range(Config['batch_size']):
                file_path = osp.join(osp.join(Config['root_path'], 'images'), B[i
+j])
                l = transforms(Image.open(file_path))
                X.append(l)
                Y.append(id_to_category[J[i+j]])

            #Y = np.stack(C)
            #Y = np.moveaxis(Y, 1, 3)

            with torch.no_grad():
                for inputs in X:
                    # print(inputs.shape)
                    inputs = inputs.to(device)
                    outputs = model_copy(inputs[None,...])
                    _, pred = torch.max(outputs,1)
                    for p in pred:
                        ans.append(p)
```

```
#             acc, loss1, ans = eval_model(model_copy_tensor, Y, criterion, devic
e)
#             # ans = (model_copy.predict(Y))

#             for k in range(len(ans)):
#                 ans1.append(np.argmax(ans[k]))
            preds = le.inverse_transform(np.asarray(ans, dtype= np.int32))
            for p in range(Config['batch_size']):
                b.write(J[p+m] + '\t' + str(preds[p]) + '\t' + id_to_category[J[p
+m]] + '\n')
            m = m + Config['batch_size']
            if m == size*Config['batch_size']:
                break
        b.close()

##############################################################################
# For Pairwise Compatibility Classification


# In[3]:


# model.py

# class MyMobileNet(nn.Module):
#     def __init__(self, my_pretrained_model):
#         super(MyMobileNet, self).__init__()
#         self.pretrained = my_pretrained_model
#         self.my_new_layers = nn.Sequential(
#                                             nn.Dropout(0.4),
#                                             nn.Linear(1000, 200),
#                                             nn.ReLU(),
#                                             nn.Linear(200, 153)
#                                             )

#     def forward(self, x):
#         x = self.pretrained(x)
#         x = self.my_new_layers(x)
#         return x

import torch.nn as nn
import torch.nn.functional as F
```

```
class Net(nn.Module):
    def __init__(self, conv1_dim=32, conv2_dim=64, conv3_dim=128, conv4_dim=256,
conv5_dim=512):
        super(Net, self).__init__()
        self.conv5_dim = conv5_dim

        self.conv1 = nn.Conv2d(3, conv1_dim, 5, stride=1, padding=2)
        self.conv2= nn.Conv2d(32, 32, 5, stride=1, padding=2)

        self.conv3 = nn.Conv2d(conv1_dim, conv2_dim, 3, stride=1, padding=2)
        self.conv4 = nn.Conv2d(conv2_dim, conv2_dim, 3, stride=1, padding=2)


        self.conv5 = nn.Conv2d(conv2_dim, conv3_dim, 3, stride=1, padding=2)
        self.conv6 = nn.Conv2d(conv3_dim, conv3_dim, 3, stride=1, padding=2)


        self.conv7 = nn.Conv2d(conv3_dim, conv4_dim, 3, stride=1, padding=2)
        self.conv8 = nn.Conv2d(conv4_dim, conv4_dim, 3, stride=1, padding=2)

        self.conv9 = nn.Conv2d(conv4_dim, conv5_dim, 3, stride=1, padding=2)
        self.conv10 = nn.Conv2d(conv5_dim, conv5_dim, 3, stride=1, padding=2)




        self.pool = nn.MaxPool2d(2, 2)
        self.dropout1=nn.Dropout(0.1)
        self.dropout2=nn.Dropout(0.2)
        self.dropout3=nn.Dropout(0.3)

        self.fc1 = nn.Linear(conv5_dim * 10 * 10, 1000) # 3x3 is precalculated an
d written, you need to do it if you want to change the # of filters
        self.fc2 = nn.Linear(1000, 500)
        self.fc3 = nn.Linear(500, 153)

        self.normalize1 = nn.BatchNorm2d(conv1_dim)
        self.normalize2 = nn.BatchNorm2d(conv1_dim)
        self.normalize3 = nn.BatchNorm2d(conv2_dim)
        self.normalize4 = nn.BatchNorm2d(conv2_dim)
        self.normalize5 = nn.BatchNorm2d(conv3_dim)
        self.normalize6 = nn.BatchNorm2d(conv3_dim)
        self.normalize7 = nn.BatchNorm2d(conv4_dim)
        self.normalize8 = nn.BatchNorm2d(conv4_dim)
        self.normalize9 = nn.BatchNorm2d(conv5_dim)
```

```
        self.normalize10 = nn.BatchNorm2d(conv5_dim)

    def forward(self, x):
        x = F.relu(self.normalize1((self.conv1(x))))
        x = self.pool(F.relu(self.normalize2((self.conv2(x))))) # first convoluti
onal then batch normalization then relu then max pool

        x = F.relu(self.normalize3((self.conv3(x))))
        x = self.pool(F.relu(self.normalize4((self.conv4(x)))))
        x=self.dropout2(x)

        x = F.relu(self.normalize5((self.conv5(x))))
        x = self.pool(F.relu(self.normalize6((self.conv6(x)))))



        x = F.relu(self.normalize7((self.conv7(x))))
        x = self.pool(F.relu(self.normalize8((self.conv8(x)))))
        x=self.dropout2(x)

        x = F.relu(self.normalize9((self.conv9(x))))
        x = self.pool(F.relu(self.normalize10((self.conv10(x)))))
        #print(x.shape)
        x = x.view(-1, self.conv5_dim * 10 * 10) # flattening the features
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x=self.dropout3(x)
        x = self.fc3(x)

        return x
model = Net()




# In[ ]:


# train_category.py

import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
```

```python
import argparse
import time
import copy
from tqdm import tqdm
import os.path as osp
import matplotlib.pyplot as plt

#from utils import Config
#from model import model
#from data import get_dataloader



def train_model(dataloader, model, criterion, optimizer, device, num_epochs, data
set_size):
    model.to(device)
    since = time.time()
    best_model_wts = copy.deepcopy(model.state_dict())
    best_acc = 0.0
    train_loss_list= []
    val_loss_list = []
    train_acc_list = []
    val_acc_list = []

    for epoch in range(num_epochs):
        print('Epoch {}/{}'.format(epoch, num_epochs - 1))
        print('-' * 10)

        for phase in ['train', 'test']:
            if phase=='train':
                model.train()
            else:
                model.eval()

            running_loss = 0.0
            running_corrects = 0

            for inputs, labels in tqdm(dataloaders[phase]):
                inputs = inputs.to(device)
                labels = labels.to(device)
                optimizer.zero_grad()

                with torch.set_grad_enabled(phase=='train'):
                    outputs = model(inputs)
                    _, pred = torch.max(outputs, 1)
```

```
                    loss = criterion(outputs, labels)

                    if phase=='train':
                        loss.backward()
                        optimizer.step()


                running_loss += loss.item() * inputs.size(0)
                running_corrects += torch.sum(pred==labels.data)

            epoch_loss = running_loss / dataset_size[phase]
            epoch_acc = running_corrects.double() / dataset_size[phase]

            if phase == 'train':
                train_loss_list.append(epoch_loss)
                train_acc_list.append(epoch_acc)

            if phase == 'test':
                val_loss_list.append(epoch_loss)
                val_acc_list.append(epoch_acc)

            print('{} Loss: {:.4f} Acc: {:.4f}'.format(phase, epoch_loss, epoch_a
cc))


            if phase=='test' and epoch_acc > best_acc:
                best_acc = epoch_acc
                best_model_wts = copy.deepcopy(model)

        #torch.save({'model':best_model_wts}, osp.join(Config['root_path'], Confi
g['checkpoint_path'], 'model.pth'))
        #print('Model saved at: {}'.format(osp.join(Config['root_path'], Config['
checkpoint_path'], 'model.pth')))
        torch.save({'model':best_model_wts}, 'model.pth')
        print('Model saved at: {}'.format('model.pth'))

    time_elapsed = time.time() - since
    print('Time taken to complete training: {:0f}m {:0f}s'.format(time_elapsed //
 60, time_elapsed % 60))
    print('Best acc: {:.4f}'.format(best_acc))

    plt.figure()
    plt.plot(np.arange(num_epochs),train_loss_list,label='Train')
    plt.plot(np.arange(num_epochs),val_loss_list, label='Validation')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
```

```
    plt.legend()
    plt.grid()
    plt.savefig('./Charu_new_loss.png', dpi=256)
    #plt.show()

    plt.figure()
    plt.plot(np.arange(num_epochs),train_acc_list,label='Train')
    plt.plot(np.arange(num_epochs),val_acc_list, label='Validation')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.legend()
    plt.grid()
    plt.savefig('./Charu_new_acc.png', dpi=256)
    #plt.show()

if __name__=='__main__':

    dataloaders, classes, dataset_size = get_dataloader(debug=Config['debug'], ba
tch_size=Config['batch_size'], num_workers=Config['num_workers'])

    criterion = nn.CrossEntropyLoss()
    optimizer = optim.RMSprop(model.parameters(), lr=Config['learning_rate'],weig
ht_decay=0.0001)
    device = torch.device('cuda:0' if torch.cuda.is_available() and Config['use_c
uda'] else 'cpu')

    train_model(dataloaders, model, criterion, optimizer, device, num_epochs=Conf
ig['num_epochs'], dataset_size=dataset_size)
    category_text_generation()

    print('DONE')
```

3. COMPATIBILTY MODEL:

```
#!/usr/bin/env python
# coding: utf-8

# utils.py

import numpy as np
import os
import os.path as osp
```

```
import argparse

Config = {}
Config['root_path'] = "./polyvore_outfits/"
Config['meta_file'] = "polyvore_item_metadata.json"
Config['test_file'] = "test_category_hw.txt"
Config['checkpoint_path'] = ''


Config['use_cuda'] = True
Config['debug'] = False
Config['num_epochs'] = 10
Config['batch_size'] = 256

Config['learning_rate'] = 0.001
Config['num_workers'] = 5  # aws might not need it, original value = 5

# In[2]:

# data.py

import torch
import torch as th
import torch.nn as nn
import torch.nn.functional as F
from torchvision import transforms
from torch.utils.data import Dataset, DataLoader
from itertools import combinations

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder

import os
import numpy as np
import os.path as osp
import json
from tqdm import tqdm
from PIL import Image


# from utils import Config


class polyvore_dataset:
    def __init__(self):
```

```
        self.root_dir = Config['root_path']
        self.image_dir = osp.join(self.root_dir, 'images')
        self.transforms = self.get_data_transforms()
        # self.X_train, self.X_test, self.y_train, self.y_test, self.classes = se
lf.create_dataset()

    def get_data_transforms(self):
        data_transforms = {
            'train': transforms.Compose([
                transforms.CenterCrop(224),
                transforms.RandomHorizontalFlip(p=0.5),
                transforms.ToTensor(),
                transforms.Normalize([0.5, 0.5, 0.5], [0.5, 0.5, 0.5])
            ]),
            'test': transforms.Compose([
                transforms.Resize(256),
                transforms.CenterCrop(224),
                transforms.RandomHorizontalFlip(p=0.5),
                transforms.ToTensor(),
                transforms.Normalize([0.5, 0.5, 0.5], [0.5, 0.5, 0.5])
            ]),
        }
        return data_transforms

    def create_dataset(self):
        meta_file = open(osp.join(self.root_dir, 'train.json'), 'r')
        meta_json = json.load(meta_file)

        ans = {}
        val_t = []
        DatAns = []

        for i in range(len(meta_json)):
            set_to_item = {}
            items = meta_json[i]["items"]
            set_id1 = meta_json[i]["set_id"]
            item_id = [sub["item_id"] for sub in items]
            # mapping set id to item-ids
            set_to_item[set_id1] = item_id
            ans.update(set_to_item)

        f = open(osp.join(self.root_dir, "compatibility_train.txt"), "r")
        f = [line.split(' ') for line in f.readlines()]
        g = f
        for i in range(len(g)):
```

```
            (g[i][len(g[i]) - 1]) = (g[i][len(g[i]) - 1]).rstrip('\n')
            if g[i][0] == '0':
                g[i].pop(1)

        for i in range(len(g)):
            lst = []
            for j in range(len(g[i])):
                if j == 0:
                    binary_label = int(g[i][j])
                    lst.append(binary_label)

                else:

                    elem = g[i][j]
                    if elem[-2] == '_':
                        set_id = elem[0:-2]
                        idx = int(elem[-1])

                    elif elem[-3] == '_':
                        set_id = elem[0:-3]
                        idx = int(elem[-2:])

                    itemid = ans[set_id][idx - 1]
                    lst.append(itemid + '.jpg')
            val_t.append(lst)

        for i in range(len(val_t)):
            for j in range(1, len(val_t[i]) - 1):
                for k in range(j + 1, len(val_t[i])):
                    flst = []
                    flst.append(val_t[i][0])
                    flst.append(val_t[i][j])
                    flst.append(val_t[i][k])
                    DatAns.append(flst)

        files = os.listdir(self.image_dir)
        Xf = []
        y = []
        for i in range(len(DatAns)):
            Xf.append(DatAns[i][1:3])
            y.append(DatAns[i][0])
        X_train, X_test, y_train, y_test = train_test_split(Xf, y, test_size=0.2)
        return X_train, X_test, y_train, y_test, max(y) + 1
```

```
# For category classification
class polyvore_train(Dataset):
    def __init__(self, X_train, y_train, transform):
        self.X_train = X_train
        self.y_train = y_train
        self.transform = transform
        self.image_dir = osp.join(Config['root_path'], 'images')

    def __len__(self):
        return len(self.X_train)

    def __getitem__(self, item):
        file_path = osp.join(self.image_dir, self.X_train[item][0])
        file_path2 = osp.join(self.image_dir, self.X_train[item][1])
        X = self.transform(Image.open(file_path))
        X2 = self.transform(Image.open(file_path2))
        return X-X2, self.y_train[item]


class polyvore_test(Dataset):
    def __init__(self, X_test, y_test, transform):
        self.X_test = X_test
        self.y_test = y_test
        self.transform = transform
        self.image_dir = osp.join(Config['root_path'], 'images')

    def __len__(self):
        return len(self.X_test)

    def __getitem__(self, item):
        file_path = osp.join(self.image_dir, self.X_test[item][0])
        file_path2 = osp.join(self.image_dir, self.X_test[item][1])
        X = self.transform(Image.open(file_path))
        X2 = self.transform(Image.open(file_path2))
        return X-X2, self.y_test[item]


def get_dataloader(debug, batch_size, num_workers):
    dataset = polyvore_dataset()
    transforms = dataset.get_data_transforms()
    X_train, X_test, y_train, y_test, classes = dataset.create_dataset()

    if debug == True:
        train_set = polyvore_train(X_train[:100], y_train[:100], transform=transf
orms['train'])
```

```
        test_set = polyvore_test(X_test[:100], y_test[:100], transform=transforms
['test'])
        dataset_size = {'train': len(y_train), 'test': len(y_test)}
    else:
        train_set = polyvore_train(X_train, y_train, transforms['train'])
        test_set = polyvore_test(X_test, y_test, transforms['test'])
        dataset_size = {'train': len(y_train), 'test': len(y_test)}

    datasets = {'train': train_set, 'test': test_set}
    dataloaders = {x: DataLoader(datasets[x],
                                 shuffle=True if x == 'train' else False,
                                 batch_size=batch_size,
                                 num_workers=num_workers)
                   for x in ['train', 'test']}
    return dataloaders, classes, dataset_size


def text_gen():
    m = 0
    le = LabelEncoder()
    #g = le.fit_transform(y)
    B = []
    f = open(Config['root_path'] + "test_pairwise_compat_hw.txt", "r")
    a = [line.split() for line in f.readlines()]
    for i in range(len(a)):
        x = [ str(a[i][0]) + '.jpg', str(a[i][1])+'.jpg']
        #print(x)
        B.append(x)

    a = open(Config['root_path'] + "test_pairwise_compat_hw.txt", "r")
    b = open('Compatibility_test_category_hw.txt', 'w')

    f = [lines.split() for lines in a.readlines()]
    J = []
    for i in range(len(f)):
        J.append(f[i][0]+ ' ' + f[i][1])

    dataset = polyvore_dataset()
    transforms = dataset.get_data_transforms()['test']

    size = int(np.floor(len(B) / Config['batch_size']))

    # model_copy_tensor = torch.load('./Results/ResNet.pth')
    check_point = torch.load('Compatibility_model.pth')
    model_copy = check_point['model']
```

```
    # model_copy = load_model('Build_model.hdf5')
    # model_copy.eval()

    for i in range(0, size * Config['batch_size'], Config['batch_size']):
        X = []
        #Y = []
        ans = []
        prob = []
        for j in range(Config['batch_size']):
            file_path = osp.join(osp.join(Config['root_path'], 'images'), B[i + j
][0])
            file_path2 = osp.join(osp.join(Config['root_path'], 'images'), B[i +
j][1])
            l = transforms(Image.open(file_path))
            l2 = transforms(Image.open(file_path2))
            X.append(l-l2)
            #Y.append(id_to_category[J[i + j]])


        with torch.no_grad():
            for inputs in X:
                # print(inputs.shape)
                inputs = inputs.to(device)
                outputs = model_copy(inputs[None, ...])
                #print(outputs)

                sm = torch.nn.Softmax(dim=1)
                probabilities = sm(outputs)
                _, pred = torch.max(outputs, 1)
                for p in pred:
                    ans.append(p)
                    prob.append(probabilities)

        #             acc, loss1, ans = eval_model(model_copy_tensor, Y, criterio
n, device)
        #             # ans = (model_copy.predict(Y))

        #             for k in range(len(ans)):
        #                 ans1.append(np.argmax(ans[k]))
        #print(type(ans))
        #preds = ans.numpy()
        for p in range(Config['batch_size']):
            answer = (ans[p].detach().cpu().numpy())
            # print(answer)
            prob_answer = np.max(prob[p].detach().cpu().numpy())
```

```
            # print(prob_answer)
            b.write(J[p + m] + '\t' + str(prob_answer) + '\t' + str(answer) + '\n
')
        m = m + Config['batch_size']
        if m == size * Config['batch_size']:
            break
    b.close()


################################################################################
# For Pairwise Compatibility Classification


# In[3]:


# model.py

from torchvision.models import resnet50
from torchvision.models import mobilenet_v2
from torch import nn

# model = resnet50(pretrained=True)
pretrained = mobilenet_v2(pretrained=True)


class MyMobileNet(nn.Module):
    def __init__(self, my_pretrained_model):
        super(MyMobileNet, self).__init__()
        self.pretrained = my_pretrained_model
        self.my_new_layers = nn.Sequential(
            nn.Dropout(0.3),
            nn.Linear(1000, 500),
            nn.ReLU(),
            nn.Linear(500, 200),
            nn.ReLU(),
            nn.Linear(200, 2)
        )

    def forward(self, x):
        x = self.pretrained(x)
        x = self.my_new_layers(x)
        return x


model = MyMobileNet(my_pretrained_model=pretrained)
```

```
# In[ ]:


# train_category.py

import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F

import argparse
import time
import copy
from tqdm import tqdm
import os.path as osp
import matplotlib.pyplot as plt


# from utils import Config
# from model import model
# from data import get_dataloader


def train_model(dataloader, model, criterion, optimizer, device, num_epochs, data
set_size):
    model.to(device)
    since = time.time()
    best_model_wts = copy.deepcopy(model.state_dict())
    best_acc = 0.0
    train_loss_list = []
    val_loss_list = []
    train_acc_list = []
    val_acc_list = []

    for epoch in range(num_epochs):
        print('Epoch {}/{}'.format(epoch, num_epochs - 1))
        print('-' * 10)

        for phase in ['train', 'test']:
            if phase == 'train':
                model.train()
            else:
                model.eval()
```

```
            running_loss = 0.0
            running_corrects = 0

            for inputs, labels in tqdm(dataloaders[phase]):
                inputs = inputs.to(device)
                labels = labels.to(device)
                optimizer.zero_grad()

                with torch.set_grad_enabled(phase == 'train'):
                    outputs = model(inputs)
                    _, pred = torch.max(outputs, 1)
                    loss = criterion(outputs, labels)

                    if phase == 'train':
                        loss.backward()
                        optimizer.step()

                running_loss += loss.item() * inputs.size(0)
                running_corrects += torch.sum(pred == labels.data)

            epoch_loss = running_loss / dataset_size[phase]
            epoch_acc = running_corrects.double() / dataset_size[phase]

            if phase == 'train':
                train_loss_list.append(epoch_loss)
                train_acc_list.append(epoch_acc)

            if phase == 'test':
                val_loss_list.append(epoch_loss)
                val_acc_list.append(epoch_acc)

            print('{} Loss: {:.4f} Acc: {:.4f}'.format(phase, epoch_loss, epoch_a
cc))

            if phase == 'test' and epoch_acc > best_acc:
                best_acc = epoch_acc
                best_model_wts = copy.deepcopy(model)

        scheduler.step()

        # torch.save({'model':best_model_wts}, osp.join(Config['root_path'], Conf
ig['checkpoint_path'], 'model.pth'))
        # print('Model saved at: {}'.format(osp.join(Config['root_path'], Config[
'checkpoint_path'], 'model.pth')))
        torch.save({'model': best_model_wts}, 'Compatibility_model.pth')
```

```
        print('Model saved at: {}'.format('Compatibility_model.pth'))

    time_elapsed = time.time() - since
    print('Time taken to complete training: {:0f}m {:0f}s'.format(time_elapsed //
 60, time_elapsed % 60))
    print('Best acc: {:.4f}'.format(best_acc))

    plt.figure()
    plt.plot(np.arange(num_epochs), train_loss_list, label='Train')
    plt.plot(np.arange(num_epochs), val_loss_list, label='Validation')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend()
    plt.grid()
    plt.savefig('./Compatibility_new_loss.png', dpi=256)
    # plt.show()

    plt.figure()
    plt.plot(np.arange(num_epochs), train_acc_list, label='Train')
    plt.plot(np.arange(num_epochs), val_acc_list, label='Validation')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.legend()
    plt.grid()
    plt.savefig('./Compatibility_new_acc.png', dpi=256)
    # plt.show()


if __name__ == '__main__':

    # root_dir = Config['root_path']
    #
    # # Parse compatibility_train.txt to whi.txt
    # meta_file = open(os.path.join(root_dir, 'train.json'), 'r')
    # meta_json = json.load(meta_file)
    # set_to_items = {}
    # for element in meta_json:
    #     set_to_items[element['set_id']] = element['items']
    #
    # file_write = open(os.path.join(root_dir, 'pairwise_compatibility_train.txt'
), 'w')
    #
    # with open(os.path.join(root_dir, 'compatibility_train.txt'), 'r') as file_r
ead:
    #     line = file_read.readline()
```

```
    #     while line:
    #         outfit = line.split()
    #         comb = list(combinations(list(range(1, len(outfit))), 2))
    #         for pair in comb:
    #             set1, idx1 = outfit[pair[0]].split('_')
    #             set2, idx2 = outfit[pair[1]].split('_')
    #             file_write.write(outfit[0] + ' ' + set_to_items[set1][int(idx1)
 - 1]['item_id'] + ' ' +
    #                              set_to_items[set2][int(idx2) - 1]['item_id'] +
 '\n')
    #
    #         line = file_read.readline()
    #
    # # Parse compatibility_valid.txt to pairwise_compatibility_valid.txt
    # meta_file = open(os.path.join(root_dir, 'valid.json'), 'r')
    # meta_json = json.load(meta_file)
    # set_to_items = {}
    # for element in meta_json:
    #     set_to_items[element['set_id']] = element['items']
    #
    # file_write = open(os.path.join(root_dir, 'pairwise_compatibility_valid.txt'
), 'w')
    #
    # with open(os.path.join(root_dir, 'compatibility_valid.txt'), 'r') as file_r
ead:
    #     line = file_read.readline()
    #     while line:
    #         outfit = line.split()
    #         comb = list(combinations(list(range(1, len(outfit))), 2))
    #         for pair in comb:
    #             set1, idx1 = outfit[pair[0]].split('_')
    #             set2, idx2 = outfit[pair[1]].split('_')
    #             file_write.write(outfit[0] + ' ' + set_to_items[set1][int(idx1)
 - 1]['item_id'] + ' ' +
    #                              set_to_items[set2][int(idx2) - 1]['item_id'] +
 '\n')
    #
    #         line = file_read.readline()

    dataloaders, classes, dataset_size = get_dataloader(debug=Config['debug'], ba
tch_size=Config['batch_size'],
                                                        num_workers=Config['num_w
orkers'])

    criterion = nn.CrossEntropyLoss()
```

```
    optimizer = optim.RMSprop(model.parameters(), lr=Config['learning_rate'], wei
ght_decay=0.0001)
    scheduler=torch.optim.lr_scheduler.StepLR(optimizer, step_size=5, gamma=0.1)
    device = torch.device('cuda:0' if torch.cuda.is_available() and Config['use_c
uda'] else 'cpu')

    train_model(dataloaders, model, criterion, optimizer, device, num_epochs=Conf
ig['num_epochs'], dataset_size=dataset_size)
    text_gen()

    print('DONE')
```